

Data processing system particularly for road vehicle engine control

Publication number: DE19748536 (A1)

Publication date: 1999-05-12

Inventor(s): LANCHES PHILLIP DR [DE]; EISENMANN JOACHIM DIPL
ING [DE]; HUBER MARTIN DIPL ING [DE]; AMINGER HANS-
JUERGEN [DE]; KOEHN MATTHIAS [DE]

Applicant(s): DAIMLER BENZ AG [DE]; IBM [US]

Classification:


- **international:** **G05B19/042; G05B19/418; G05B19/04; G05B19/418;** (IPC1-
7): G05B19/042


- **European:** G05B19/042M; G05B19/418N

Application number: DE19971048536 19971103

Priority number(s): DE19971048536 19971103

Also published as:

 DE19748536 (C2)

 US6336128 (B1)

Abstract of **DE 19748536 (A1)**

The data processing system is configured as a number of processing modules (1), each of which handle different vehicle functions. The modules are connected via a data processing network (2). The application functions are in the form of a client server architecture that has client, server and function monitor levels.

.....
Data supplied from the **esp@cenet** database — Worldwide



⑮ **BUNDESREPUBLIK
DEUTSCHLAND**



**DEUTSCHES
PATENT- UND
MARKENAMT**

⑫ **Offenlegungsschrift**
⑩ **DE 197 48 536 A 1**

⑤① Int. Cl.⁶:
G 05 B 19/042

②① Aktenzeichen: 197 48 536.7
②② Anmeldetag: 3. 11. 97
④③ Offenlegungstag: 12. 5. 99

DE 197 48 536 A 1

⑦① **Anmelder:**
Daimler-Benz Aktiengesellschaft, 70567 Stuttgart,
DE; International Business Machines Corp.,
Armonk, N.Y., US

⑦④ **Vertreter:**
Rach, W., Dr., Pat.-Ass., 70569 Stuttgart

⑦② **Erfinder:**
Lanchès, Phillip, Dr., 73728 Esslingen, DE;
Eisenmann, Joachim, Dipl.-Ing., 73079 Süßen, DE;
Huber, Martin, Dipl.-Ing., 70806 Kornwestheim, DE;
Aminger, Hans-Jürgen, 73061 Ebersbach, DE;
Köhn, Matthias, 72555 Metzingen, DE

⑤⑥ **Entgegenhaltungen:**
KUSCHKE, D. Softwareschnittstelle
SPS-INTERBUS-S. In: Elektrische Berlin 48
(1994) 1, S.26-30;
R. ORFALI, D. HARKEY: Client/Server Pro-
gramming with OS/2 2.0. Van NOSTRAND Rein-
hold, New York 1992. S.10-14 u. 165-167;

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

Prüfungsantrag gem. § 44 PatG ist gestellt

⑤④ **Datenverarbeitungsgestütztes elektronisches Steuerungssystem, insbesondere für ein Kraftfahrzeug**

⑤⑦ Die Erfindung bezieht sich auf ein Steuerungssystem mit einem Anwendungsfunktionen ausführenden Steuergeräteverbund mit mehreren, verteilt angeordneten Steuergeräten und einem diese verbindenden Datenübertragungsnetzwerk.
Erfindungsgemäß sind die Anwendungsfunktionen in Form einer Client/Server-Architektur in dem Steuergeräteverbund implementiert. Dies ermöglicht die Durchführung von Anwendungsfunktionen in Echtzeit mittels eines flexiblen, standardisierten und offenen Systems, mit dem Änderungen und/oder Ergänzungen von Anwendungsfunktionen mit relativ geringem Aufwand realisierbar sind.
Verwendung z. B. als Steuerungssystem in einem Kraftfahrzeug.

DE 197 48 536 A 1

Die Erfindung bezieht sich auf ein datenverarbeitungs-
gestütztes elektronisches Steuerungssystem mit einem An-
wendungsfunktionen ausführenden Steuergeräteverbund
mit mehreren, verteilt angeordneten Steuergeräten und ei-
nem diese verbindenden Datenübertragungsnetzwerk. Der
Begriff Steuerungssystem ist hierbei in seinem weiteren,
auch Regelungssysteme umfassenden Sinne zu verstehen.

Derartige Steuerungssysteme werden beispielsweise in
Kraftfahrzeugen eingesetzt, um dort die fahrzeugtypischen
Steuerungsfunktionen auszuführen. In herkömmlichen Sys-
temen werden die Steuergeräte jeweils spezifisch für eine
oder mehrere Anwendungsfunktionen ausgelegt. Die Imple-
mentierung einer neuen Fahrzeugfunktion erfordert den Ent-
wurf eines zugehörigen neuen Steuergerätes und dessen
Einbau im Fahrzeug zusammen mit einer neuen Sensor- und
Aktuatorkonfiguration. Zwar sind die Steuergeräte in mo-
dernerer Konfigurationen z. B. über einen CAN-Bus ver-
netzt, jedoch existieren keine expliziten Schnittstellen für
den Zugriff auf einzelne Funktionsteile, so daß die jeweilige
Anwendungsfunktion aus Sicht des Steuergerätes nur als
Ganzes in Erscheinung tritt. Zur Realisierung von neuen,
auf existierende Funktionen aufgebauten, sogenannten re-
kombinierten Funktionen müssen diese folglich mit hohem
Aufwand von Hand an existierende Funktionen angeschlossen
werden, z. B. durch Definition oder Änderung entspre-
chender CAN-Botschaften. In ungünstigen Fällen macht
dies für die Hinzufügung einer einzelnen neuen Funktion
die Änderung aller anderen Steuergeräte nötig. Dies führt
zusammen mit dem Trend zu immer mehr elektronisch rea-
lisierten Funktionen im Kraftfahrzeug und deren zunehmen-
der Kopplung untereinander zu einer stark anwachsenden
Komplexität und zu entsprechenden Schwierigkeiten bei der
Entwicklung und Beherrschung der gesamten Fahrzeugelek-
tronik sowie zu einem steigenden Bedarf an Rechenlei-
stung und Speicherumfang. Zudem ergibt sich durch die
steigende Komplexität bei gleichzeitig immer mehr Baurei-
hen und kürzeren Entwicklungszeiten für dieselben der Be-
darf, Komponenten vermehrt baureihenübergreifend wie-
derverwenden zu können.

Von reinen Datenverarbeitungssystemen, z. B. zur Büro-
kommunikation und in Großrechenanlagen, mit verteilten
Rechnerkapazitäten ist es bekannt, sogenannte Client/Ser-
ver-Architekturen vorzusehen, um einen jeweiligen Dienst
von einem hierauf ausgelegten Server für diesen Dienst auf-
rufende Clients zentralisiert zu erbringen. Bei diesen Systeme
ist im allgemeinen keine Echtzeitverarbeitung gegeben.

Der Erfindung liegt als technisches Problem die Bereit-
stellung eines insbesondere auch für Kraftfahrzeuge an-
wendbaren Steuerungssystems der eingangs genannten Art
zugrunde, das sich zur Durchführung der Anwendungsfunk-
tionen in Echtzeit und auch mit relativ knappen Rechenka-
pazitäten eignet und möglichst flexibel, standardisiert und
offen ausgelegt ist, um Änderungen und/oder Ergänzungen,
insbesondere hinsichtlich Implementierung neuer und/oder
Modifikationen bestehender Anwendungsfunktionen, mit
vergleichsweise geringem Aufwand realisieren zu können.

Die Erfindung löst dieses Problem durch die Bereitstel-
lung eines Steuerungssystems mit den Merkmalen des An-
spruchs 1. Bei diesem datenverarbeitungsgestützten elektro-
nischen Steuerungssystem sind die vom System durchzu-
führenden Anwendungsfunktionen in Form einer Client/
Server-Architektur in den Steuergeräteverbund implemen-
tiert. Die Client/Server-Architektur wird vorliegend in ei-
nem sogenannten Embedded-System verwendet, d. h. einem
System, in welchem die elektronische Datenverarbeitungs-
funktionalität in ein übergeordnetes System, wie z. B. eine

Fahrzeugfunktionen ausführende Fahrzeugelektronik, die-
ses stützend eingebettet ist und dem Anwender nicht direkt
in Erscheinung tritt. Durch die Übertragung der aus Daten-
verarbeitungssystemen bekannten Client/Server-Architek-
tur auf das vorliegende Steuerungssystem wird ein Modell
zur Strukturierung verteilter Anwendungen bereitgestellt,
das besonders gut zur Beschreibung ereignisorientierter Sys-
teme geeignet ist, wobei im Unterschied zu konventionel-
len Systemen die Schnittstellen zwischen Client- und Ser-
ver-Prozessen primär an Diensten und nicht an Daten orien-
tiert sind. Mit dem vorliegenden System können die Anwen-
dungsfunktionen hardwareunabhängig entwickelt und rela-
tiv einfach zwecks Erzeugung neuer, höherwertiger Anwen-
dungsfunktionen rekombiniert werden. Im Rahmen der
Client/Server-Architektur werden die Anwendungsfunktionen
beschrieben, die über definierte Anwendungsprotokoll-
Schnittstellen miteinander kommunizieren, wozu zum Ent-
wurfszeitpunkt noch keine Aussage über die Art der physika-
lischen Kommunikation gemacht wird. Damit lassen sich
durch das erfindungsgemäße Steuerungssystem Anwen-
dungsfunktionen in Echtzeit durchführen und vergleichs-
weise einfach in verschiedene Systemauslegungen, z. B. bei
verschiedenen Fahrzeugbaureihen, implementieren. Die
elektronische Infrastruktur des Steuerungssystems besitzt
durch die Verwendung des Client/Server-Architekturmo-
dells eine flexible, standardisierte und offene Basis und eig-
net sich insbesondere auch für Systeme mit vergleichsweise
geringen Rechenleistungsressourcen und statischer Soft-
ware-Konfiguration.

Bei einem nach Anspruch 2 weitergebildeten Steuerungs-
system beinhaltet die Client/Server-Architektur für eine je-
weilige Anwendungsfunktion eine zwischen einer Client-
ebene und einer Serverebene liegende Funktionsmonitor-
ebene, die den globalen Zustand der Anwendungsfunktion
verwaltet und somit als deren zentrale Schaltstelle oder Gedächtnis fungiert. In weiterer Ausgestaltung dieser Strukturi-
erung sind nach Anspruch 3 eine oder mehrere dieser drei
Ebenen in spezieller Weise weiter strukturiert. Die Client-
ebene kann hierbei aus Requestoren als Quellen von Dienst-
anforderungen und nachgeschalteten primären Clients be-
stehen, während analog dazu die Serverebene aus primären
Servern und nachgeschalteten Fulfillern aufgebaut sein
kann. Ein primärer Client und der zugehörige Requestor
bzw. ein primärer Server und der zugehörige Fulfiller wer-
den stets auf demselben Steuergerät angeordnet. Die Moni-
torebene beinhaltet einen mit den primären Clients kommu-
nizierenden Funktionsmonitor, der den globalen Zustand der
Anwendungsfunktion verwaltet, und von diesem abhängige
Monitore für die Verwaltung von Teilfunktionen.

Der Funktionsentwurf basiert auf einer Menge von defi-
nierten Designelementen wie Methoden und Protokolle,
Service-Access-Points und Service-Access-Point-Inter-
faces, Ports und Portinterfaces, Verbindungen, Prozessen,
Frames und Firmwareprozessen. In Weiterbildung der Erfin-
dung nach Anspruch 4 sind die Service-Access-Points cha-
rakteristischerweise so ausgelegt, daß sie Schnittstellen von
Anwendungsprozessen zur Schicht 7 des ISO/OSI-Refe-
renzmodells bilden und je ein Protokoll in Client- und in
Serverrolle enthalten. In einer Ausgestaltung der Erfindung
nach Anspruch 5 sind die Ports als Ankerpunkte für bidirek-
tionale Client/Server-Kommunikationsverbindungen zur
Implementierungszeit ausgelegt und stellen eine horizontale
Kommunikations-Schnittstelle auf der Schicht 7 des ISO/
OSI-Referenzmodells dar.

In weiterer Ausgestaltung der Erfindung nach Anspruch 6
sind die Prozesse als Designelemente, welche die eigentli-
che Anwendungssoftware beinhalten, in spezieller Weise
aus einer äußeren Schnittstelle, einer inneren Struktur und

einem spezifischen Verhalten auf eingehende Dienst Anforderungen aufgebaut.

In Weiterbildung der Erfindung nach Anspruch 7 besitzt das Steuerungssystem ein echtzeitfähiges Multitasking-Betriebssystem in den Steuergeräten, und die Client/Server-Prozesse sind so implementiert, daß sie ohne direkten Hardwarezugriff nur die Dienste des Betriebssystems und einer zugehörigen Kommunikationsschicht nutzen, die auf der sogenannten Remote-Procedure-Call (RPC)-Technik basiert, wie sie der Art nach von Datenverarbeitungssystemen mit Client/Server-Architektur für eine solche Kommunikation zwischen Client/Server-Prozessen bekannt ist.

In weiterer Ausgestaltung der Erfindung ist nach Anspruch 8 vorgesehen, daß in einer entsprechenden RPC-Bibliothek ein vollständiger Server-Code enthalten ist, der pro Steuergerät genau einmal eingebunden ist und von allen Client- und Server-Prozessen abgearbeitet wird. Dies minimiert den Ressourcenbedarf und maximiert gleichzeitig die Wiederverwendungsfähigkeit.

In weiterer Ausgestaltung der Erfindung erfolgt die Initialisierung des Servers gemäß Anspruch 9 in vier speziellen Phasen.

Gemäß einer Weiterbildung der Erfindung nach Anspruch 10 ist der Remote-Procedure-Call (RPC) als synchroner RPC, asynchroner RPC oder Oneway-RPC realisiert.

In Weiterbildung der Erfindung nach Anspruch 11 ist eine minimale bzw. ressourcenoptimierte Protokollimplementierung vorgesehen, die jeder Methode eine identifizierende Dienstnummer zuweist und bei der in den Nutzdaten der versendeten Nachrichten sowohl der Nachrichtentyp als auch die Dienstnummer und die übergebenden Daten codiert sind. Das so implementierte Protokoll ist z. B. auf einem CAN-Bus verwendbar.

Vorteilhafte Ausführungsformen der Erfindung sind in den Zeichnungen dargestellt und werden nachfolgend beschrieben. Hierbei zeigen:

Fig. 1 eine ausschnittsweise Blockdiagramm-Darstellung eines Steuergeräteverbundes eines Kraftfahrzeug-Steuerungssystems,

Fig. 2 eine Blockdiagramm-Darstellung der Entwurfsstruktur einer im Steuerungssystem von **Fig. 1** implementierten Client/Server-Architektur,

Fig. 3 eine Darstellung des graphischen Erscheinungsbildes der inneren Struktur einer Prozeßklasse für die Client/Server-Architektur,

Fig. 4 eine graphische Darstellung der Einbettung eines endlichen Automaten in eine Prozeßklasse der Client/Server-Architektur,

Fig. 5 eine graphische Darstellung des Client/Server-Designs am Beispiel einer Rückfahrscheinwerfer-Anwendungsfunktion,

Fig. 6 ein Blockdiagramm der für die Client/Server-Architektur verwendeten Steuergeräte-Software,

Fig. 7 eine blockdiagrammatische Darstellung der Prozeßkommunikation der Client/Server-Architektur mittels Protokollen auf Anwendungsschichtebene,

Fig. 8 ein Blockdiagramm eines prinzipiellen ORPC-Ablaufs in der Client/Server-Architektur,

Fig. 9 ein Flußdiagramm des Server-Arbeitsablaufs in der Client/Server-Architektur,

Fig. 10 ein Flußdiagramm des Ablaufs einer Server-Initialisierungsphase,

Fig. 11 ein Flußdiagramm einer synchronen RPC-Implementierung,

Fig. 12 ein Flußdiagramm einer Oneway-RPC-Implementierung und

Fig. 13 eine schematische Blockdiagramm-Darstellung des implementierten Protokolls der Client/Server-Architektur.

tur.

Fig. 1 zeigt ausschnittsweise und blockdiagrammatisch mehrere, verteilt angeordnete Steuergeräte 1a, 1b, 1c und ein diese verbindendes Datenübertragungsnetzwerk mit einem Datenbus 2, z. B. einem CAN-Bus, eines Steuergeräteverbunds zur Ausführung von Anwendungsfunktionen in einem Kraftfahrzeug. In dem Steuergeräteverbund dieses Steuerungssystems sind die Anwendungsfunktionen in Form einer Client/Server-Architektur (CSA) implementiert, die dafür sorgt, daß alle Systemschnittstellen beschrieben sind und nur über selbige mit den Objekten kommuniziert wird und Daten ausgetauscht werden. Soweit diese Client/Server-Architektur in ihrer Struktur und ihren Komponenten den herkömmlichen Client/Server-Architekturen entspricht, wird darauf im folgenden nicht näher eingegangen, sondern auf die betreffende Literatur verwiesen und die entsprechende Terminologie verwendet. Durch diese Architektur wird eine Portabilität der Software zwischen verschiedenen Hardware-Plattformen erreicht. Die Nutzung eines von herkömmlichen objektorientierten Methoden bekannten Construction-from-Parts-Ansatzes kann dafür sorgen, daß sich einmal spezifizierte, implementierte und getestete Programme immer wieder verwenden lassen. Weiter wird durch Realisierung der Kommunikation auf Basis eines Remote-Procedure-Calls (RPC) gewährleistet, daß einzelne Prozesse beliebig im Steuergerätenetzwerk verteilt werden können, ohne die Funktionsentwürfe oder die implementierte Software manuell anpassen zu müssen. **Fig. 1** veranschaulicht beispielhaft, wie auf diese Weise eine Applikation aus einem Client-Prozeß 3, einem Funktionsmonitor 3b und einem Server-Prozeß 4a, 4b auf den Steuergeräten 1a, 1b, 1c verteilt werden kann. Ein weiterer Server 4a gehört nicht zu dieser Client/Server-Kette. Der Funktionsmonitor 3b fungiert sowohl als Server (gegenüber Client 3) wie auch als Client (gegenüber Server 4b).

Fig. 2 zeigt im Blockdiagramm den Systementwurf einer jeweiligen Anwendungsfunktion im Rahmen der erfindungsgemäß verwendeten Client/Server-Architektur. Gemäß diesem Systementwurf ist die jeweilige Anwendungsfunktion in eine Clientebene 5, eine Serverebene 6 und eine zwischenliegende Funktionsmonitorebene 7 strukturiert. Die Clientebene 5 beinhaltet einen oder mehrere primäre Clients 5a mit vorgeschaltetem Requestor 5b. Die Requestoren 5b repräsentieren ereignisauslösende Hardwareeinheiten, wie Sensoren, und die zugehörige Steuergeräte-Firmware. Sie stellen die Quellen von Dienst Anforderungen der modellierten Anwendungsfunktion dar. Die primären Clients 5a verwalten die Requestoren, nehmen deren Dienst Anforderungen entgegen und setzen gegebenenfalls weitere Aufträge an die Funktionsmonitorebene 7 ab. Die primären Clients 5a werden stets auf demselben Steuergerät wie der zugehörige Requestor 5b angesiedelt und erlauben es, die Dienst Anforderung auch über Kommunikationsmedien weiterzuleiten, wozu der Requestor 5b nicht ausgelegt ist. Die Funktionsmonitorebene 7 beinhaltet für jede Anwendungsfunktion einen Funktionsmonitor 7a, der die Dienst Anforderungen von primären Clients 5a und gegebenenfalls von Funktionsmonitoren übergeordneter Anwendungsfunktionen entgegennimmt und bearbeitet. Er kann dazu weitere Dienste von ihm untergeordneten Monitoren oder primären Servern in Anspruch nehmen. Der Funktionsmonitor 7a verwaltet den globalen Zustand der Anwendungsfunktion und bildet damit deren zentrale Schaltstelle und Gedächtnis. Dem Funktionsmonitor 7a sind innerhalb der Funktionsmonitorebene 7 gegebenenfalls ein oder mehrere Monitore 7b nachgeschaltet, die Teilfunktionen verwalten und sich vom Funktionsmonitor 7a dadurch unterscheiden, daß sie nicht direkt mit primären Clients 5a und Funk-

tionsmonitoren 7a anderer Anwendungsfunktionen kommunizieren. Die Serverebene 6 beinhaltet einen oder mehrere primäre Server 6a und einen jeweils zugehörigen Fulfiller 6b. Die Fulfiller 6b repräsentieren ausführende Hardware-Einheiten, wie Aktuatoren, und die zugehörige Steuergeräte-Firmware. Sie stellen die Senken von Dienstanforderungen der modellierten Anwendungsfunktion dar. Die primären Server 6a verwalten die Fulfiller 6b und beauftragen diese mit der Ausführung von Diensten. Sie sind stets auf demselben Steuergerät wie der zugehörige Fulfiller 6b angesiedelt und erlauben es, Dienstanforderungen von entfernten Monitoren 7b entgegenzunehmen, worauf die Fulfiller 6b nicht ausgelegt sind.

Die Pfeile in Fig. 2 repräsentieren Client/Server-Beziehungen, wobei der Pfeil vom Client zum jeweiligen Server verläuft und für ein bestimmtes Anwendungsprotokoll steht. Normalerweise verhält sich ein Server dabei synchron zu den aufrufenden Clients. Die Richtung der Pfeile deutet auf diese Betriebsart hin. In Ausnahmesituationen ist es jedoch auch möglich, daß ein Server 6a asynchron Informationen an einen oder mehrere seiner Clients 5a sendet. In diesem Ausnahmebetrieb findet eine Umkehr der jeweiligen Rollen statt, für die ein eigenes Protokoll zu vereinbaren ist.

Der Funktionsentwurf für die Client/Server-Architektur mit der in Fig. 2 dargestellten Anwendungs-Entwurfsstruktur basiert auf einer Menge von hierfür geeignet definierten Designelementen. Die Entwurfsmethodik sieht dabei eine spezielle graphische Notation für die Designelemente vor, wodurch sie von graphischen Entwurfswerkzeugen unterstützt werden kann, was die Lesbarkeit von Entwürfen sowie das Erlernen der Methodik verglichen mit rein textuellen Notationen erleichtert. Speziell sind als Designelemente den Requestoren 5b eine Sensor-Firmware, den Fulfillern 6b eine Aktuator-Firmware, den Client/Server-Beziehungen Verbindungen und den primären Clients 5a, den Funktionsmonitoren 7a, den Monitoren 7b und den primären Servern 6a Prozeßklassen zugeordnet.

Allgemein sind als Designelemente Methoden und Protokolle, Service-Access-Points (SAP) und Service-Access-Point-Interfaces (SAPIF), Ports und Port-Interfaces (PortIF), Verbindungen, Datenelemente, Prozesse, Frames und Firmware-Prozesse vorgesehen. Methoden stellen in diesem Zusammenhang Funktionalität dar, die eine Prozeßklasse anderen Prozeßklassen als Dienste an den SAPs in Serverrolle zur Verfügung stellt. Andererseits kann dieselbe Prozeßklasse an SAPs in Clientenrolle Dienste anfordern, die von anderen Prozeßklassen als Methoden an SAPs in Serverrolle angeboten werden. Methoden haben wie herkömmliche Funktionsaufrufe eine Typ und Argumente, wobei der Typ angibt, welches Datenformat der Rückgabewert der Methode hat. Für die Anwendung ist es transparent, ob die Dienstanforderung als Remote-Procedure-Call mittels Senden/Empfangen über ein externes physikalisches Kommunikationsmedium, mittels Interprozeßkommunikation oder mittels Eventmechanismus vom Client zum Server kommuniziert wird. Protokolle dienen der Aggregation von Methoden. Zu diesem Zweck beinhalten sie eine Liste von Methoden, die aus Sicht des Anwendungsentwicklers eine funktionale Einheit darstellen. Des weiteren können Protokolle hierarchisch strukturiert werden, d. h. es kann eine Hierarchie von Protokollen dergestalt aufgebaut werden, daß ausgehend von einem Null-Protokoll, das keine Methoden enthält, eine Baumstruktur von Protokollen entsteht, in der ein neu hinzugefügtes Protokoll alle Methoden des zur Wurzel des Baumes hin nächstliegenden Protokolles erbt und gegebenenfalls weitere hinzufügt.

Die SAPs sind die Schnittstellen von Anwendungsprozessen zur Schicht 7 des ISO/OSI-Referenzmodells und bein-

halten je ein Protokoll in Client- und in Serverrolle. Die SAPs haben eine Vorzugsrichtung, die entweder der Client- oder der Serverrolle entspricht. Da SAPs Endpunkte von mehreren Client/Server-Kommunikationsverbindungen sein können, beinhalten sie eine entsprechende Anzahl an Ports, die der Verankerung der einzelnen Kommunikationsverbindungen dienen. SAPIFs machen SAP-Funktionalität an Prozeß- und Frameklassen-Schnittstellen verfügbar, wobei ein SAP innerhalb einer Prozeßklasse Verbindungen zu mehreren SAPIFs haben kann. Dies bietet unter anderem die Möglichkeit, den SAPIFs entsprechende Namen zu geben, z. B. Rückfahrlicht rechts, Rückfahrlicht links, und damit den gezielten Anschluß weiterer Komponenten zu ermöglichen, obwohl die Funktionalität von einem einzigen SAP erfüllt wird. SAPIFs existieren nur während des hierarchischen Systementwurfs. Nach Auflösung der Hierarchie vor der Instanzierung besteht keine Notwendigkeit mehr, diesem Designelement eine Entsprechung in der Implementierung zu geben.

Ports sind Ankerpunkte für bidirektionale Client/Server-Kommunikationsverbindungen zur Implementierungszeit. In dieser Eigenschaft stellen sie eine horizontale Kommunikationsschnittstelle auf Schicht 7 des ISO/OSI-Referenzmodells dar. An den Ports wird der eigentliche Kommunikationsmechanismus verankert, d. h. das Senden/Empfangen über externe physikalische Kommunikationsmedien, die Interprozeßkommunikation und ein Eventmechanismus für die effektive Kommunikation mit Firmware-Prozessen. Von einem SAP ausgehende Dienstanforderungen können folglich über unterschiedliche Kommunikationsmechanismen weitergeleitet werden, je nachdem, mit welchem Kommunikationsmechanismus der jeweils gerade angesprochene Port hinterlegt ist. Analog zu den SAPIFs machen die PortIFs Port-Funktionalität an Schnittstellen von nachfolgend beschriebenen Prozeß- und Frameklassen verfügbar. Die PortIFs gehören entweder zur Schnittstelle eines gerade entworfenen Frames oder zur Schnittstelle eines darin enthaltenen Frames oder Prozesses.

Jede Prozeßklasse kann Datenelemente enthalten, welche sie im Sinne der Objektorientierung kapselt. Ein Datenelement hat einen Datentyp und kann durch einen Initialisierungswert vorbelegt werden. Es kann konstant oder variabel sein und ist dementsprechend in einem ROM oder RAM angeordnet. Ferner werden die Datenelemente als private oder public klassifiziert. Die Initialisierung von private-Datenelementen hat direkt beim Entwurf der Prozeßklasse zu erfolgen, während diejenige von public-Datenelementen auf einer hierarchisch höheren Designebene geschieht, wenn der entsprechende Kontext bekannt ist, in welchem die Prozeßklasse verwendet wird. Die in der Prozeßinstanz gekapselten Datenelemente können nur von der Prozeßinstanz selbst geändert werden.

Ein Prozeß im Rahmen der vorliegenden Client/Server-Architektur hat Schnittstellen, über die er mit anderen Prozessen oder Frames verbunden werden kann. Er bietet Dienste an SAPIFs in Serverrolle an und nutzt Dienste von anderen Prozessoren an SAPIFs in Clientrolle. Das Verhalten einer Prozeßklasse wird mittels endlicher Automaten beschrieben. Jede Prozeßklasse kann als Summe dreier Komponenten betrachtet werden, und zwar einer äußeren Schnittstelle, einer inneren Struktur und des Verhaltens. Fig. 3 zeigt ein Beispiel des graphischen Erscheinungsbildes der inneren Struktur einer Prozeßklasse mit einer zugehörigen äußeren Schnittstelle. Das Verhalten einer Prozeßklasse als Reaktion auf eine eingehende Dienstanforderung gliedert sich in drei Teilaspekte, und zwar in die Änderung des inneren Zustands z der Prozeßklasse, d. h. eines endlichen Automaten FSM, der das Verhalten der Prozeßklasse repräsentiert.

tiert, in die Modifikationen an den gekapselten Datenelementen (process data set) und die Ausführung von Aktionen und gegebenenfalls Wechsel des Prozesses in Clientrolle, um Dienste von anderen Prozessen in Anspruch zu nehmen. **Fig. 4** zeigt die Einbettung des endlichen Automaten, d. h. der Finite State Machine (FSM), in die Prozeßklasse. Wie aus **Fig. 4** erkennbar, können eingehende Dienstanforderungen sowohl den x-Vektor des endlichen Automaten als auch den Process Data Set beeinflussen. Ändert sich der x-Vektor, wird gemäß der Definition des endlichen Automaten geprüft, ob gegebenenfalls ein neuer Zustand erreicht wird und entsprechende Aktionen ausgeführt werden müssen. Welche Aktionen ausgeführt werden, ergibt sich aus der Interpretation des y-Vektors. Das Ausführen der Aktionen kann weitere Änderungen am Process Data Set und den Wechsel in die Clientrolle mit Anforderung von Diensten anderer Prozesse bedeuten. Weiter ist der **Fig. 4** die charakteristische Maßnahme entnehmbar, die SAPs in Server- und Client-Modus auf Anwendungsebene zu implementieren.

Um einen hierarchischen Softwareentwurf zu ermöglichen, sind Frames als weitere Designelemente vorgesehen, die in ihrer internen Struktur weitere Frames, Prozesse und notwendige Verbindungen kapseln können. Ein Frame hat wie ein Prozeß Schnittstellen, über die er mit anderen Prozessen oder Frames verbunden werden kann. Er bietet Dienste an SAPIFs in Serverrolle an und nutzt Dienste von anderen Prozessen an SAPIFs in Clientrolle. Prinzipiell können Frames zusätzlich mit einer Verhaltensbeschreibung versehen sein, was schließlich ein Zusammenfallen mit Prozessen bedeuten kann, so daß nur noch eines dieser beiden Designelemente auf Designebene benötigt wird. In diesem Fall müssen für die Implementierung einer Anwendung für alle Prozeßklassen auf hierarchisch unterster Designebene explizite Verhaltensbeschreibungen existieren, während dies für Prozeßklassen auf hierarchisch höheren Ebenen optional ist, wobei dann deren Verhaltensbeschreibung mit derjenigen der in ihr enthaltenen Prozeßklassen niedrigerer Ebene zusammenpassen muß.

Als weiteres Designelement sind Firmware-Prozesse vorgesehen, mit denen Firmware charakteristischerweise als Prozesse beschrieben wird und die das Bindeglied zwischen einer jeweiligen Hardwarekomponente und genau einem zugeordneten primären Client oder Server bilden. Sie kommunizieren mit letzteren über Anwendungsprotokolle, wie sie auch zur Kommunikation zwischen den Prozessen verwendet werden und besitzen demzufolge ebenfalls mindestens ein SAPIF. Im Unterschied zu Prozessen werden jedoch weder innere Struktur, noch Verhalten von Firmware-Prozessen beschrieben, es können jedoch Datenelemente verwendet werden.

Fig. 5 zeigt am Beispiel der typischen fahrzeugspezifischen Anwendungsfunktion "Rückfahrscheinwerfer" beispielhaft die prinzipielle Vorgehensweise beim Funktionsentwurf gemäß dem vorliegenden Client/Server-Modell. Die graphische Darstellung von **Fig. 5** zeigt hierzu ein äußeres Frame **8**, das zwei innere Frames **9, 10** und einen durch die Graphik seiner äußeren Schnittstelle **11** dargestellten Prozeß beinhaltet. Im ersten Schritt erfolgt die Identifikation der beteiligten Sensorik und Aktuatorik und damit die Festlegung der primären Clients und Server. Im gezeigten Beispiel wird die Funktion durch einen Schalter an der Schaltkulisie eines Automatgetriebes aktiviert. Als Aktuator wird am Fahrzeugheck eine Glühlampe eingeschaltet. Als Endpunkte dieses Client/Server-Szenarios ergeben sich damit ein primärer Client zur Überwachung des Rückfahrschalters und ein primärer Server zur Ansteuerung der digitalen Lampenendstufe für den Rückfahrscheinwerfer. Die eigentliche Funktionslogik befindet sich in einem Monitor, der in die-

sem Fall lediglich den Schaltzustand mit der Zündklemmeninformation verknüpft, so daß der Rückfahrscheinwerfer bei ausgeschalteter Zündung nicht eingeschaltet wird.

Nachdem auf diese Weise die Struktur der Funktion festliegt, werden im nächsten Schritt die Protokolle zwischen den Clients und Servern definiert. Ein Protokoll besteht dabei aus einer Anzahl von Diensten, die der Server dem Client anbietet. Im Beispielfall eines binären Schalters ist zwischen primärem Client und Monitor nur die Information "Schalter eingeschaltet" oder "Schalter ausgeschaltet" auszutauschen. Bei der Aktuatorikansteuerung ergeben sich ebenfalls nur zwei Dienste, die der primäre Server dem Funktionsmonitor anbieten muß, nämlich "Endstufe einschalten" und "Endstufe ausschalten".

Ein Blick in eine während der Entwicklung entstandene Parts-Bibliothek kann zeigen, daß bereits ein Protokoll zur Übertragung binärer Information existiert, das im Beispielfall sowohl für die Kommunikation zwischen primärem Client und Monitor als auch zwischen Monitor und primärem Server genutzt werden kann. Die entsprechenden Prozesse und Protokolle zur Einspeisung der Zündklemmeninformation in dieses Funktionsszenario sind ebenfalls bereits in der Parts-Bibliothek enthalten, wobei in diesem Fall eine Kommunikationsverbindung zu demjenigen Client genügt, der die aktuelle Zündklemme an den Funktionsmonitor weitergibt.

Nachfolgend wird detaillierter auf die Implementierung der Client/Server-Architektur im Fahrzeug eingegangen. Eine grundlegende Komponente bildet das verwendete Betriebssystem. Um die Portabilität der Client/Server-Prozesse auf unterschiedliche Hardware-Plattformen sicherzustellen, ist auf den Steuergeräten eine Betriebssystemschicht realisiert, welche die Abhängigkeiten der Software zur Hardware kapselt und eine abstrakte Programmierschnittstelle (Application Programming Interface; API) bereitstellt. Voraussetzung für den Einsatz der Client/Server-Architektur in einem Steuergerät ist ein echtzeitfähiges Multitasking-Betriebssystem. Beispielsweise kann das Echtzeit-Betriebssystem OSEK mit der Conformance-Class ECC1 gewählt werden, da für die Kommunikation der Event-Mechanismus notwendig ist. Die gegenseitige Unterbrechbarkeit einzelner Tasks, d. h. preemptives Scheduling, ist nicht zwingend erforderlich. **Fig. 6** zeigt die Struktur von Software-Modulen auf einem Steuergerät mit der vorliegenden Client/Server-Architektur sowie die Koexistenz zu konventionellen Applikationen, wobei zusätzlich die Schichten des ISO/OSI-Referenzmodells zu Vergleichszwecken dargestellt sind. Aus **Fig. 6** ist ersichtlich, daß die Client/Server-Prozesse nicht direkt auf die Hardware zugreifen, sondern die Dienste des Betriebssystems und der ORPC-Kommunikationsschicht und damit des OSEK COM nutzen, wobei die Kommunikationsbeziehungen zwischen Client/Server-Prozessen mit gestrichelten Linien wiedergegeben sind. Die gebogene Linie repräsentiert eine Client/Server-Kommunikation auf demselben Gerät, während die gerade Linie eine solche zwischen verschiedenen Geräten symbolisiert. Durch diese Struktur läßt sich eine Verschiebbarkeit der Client/Server-Anwendung zwischen verschiedenen Steuergeräten erreichen, die alle über entsprechende OSEK OS-, OSEK COM- und ORPC-OXDR-Implementierungen verfügen. In **Fig. 7** ist die vorliegend getroffene Maßnahme veranschaulicht, die SAPs und die Ports für das Zusammenspiel von Prozessen mittels Protokollen auf dem Niveau von Schicht 7 des ISO/OSI-Referenzmodells zu implementieren.

Wie gesagt, greifen die Client/Server-Anwendungen auf Dienste einer Kommunikationsschicht zurück. Dabei setzt die ORPC(OSEK-basierter Remote-Procedure-Call)-Schicht auf den Timer- und Event-Diensten des Betriebssy-

stems und den Kommunikationsroutinen der OSEK COM-Ebene auf. Die OSEK COM-Schicht stellt Kommunikationsroutinen bereit, über die Tasks miteinander Daten austauschen können. Die Kommunikation zweier Tasks erfolgt konfigurationsabhängig innerhalb desselben Adressraums oder zwischen verschiedenen Adressräumen über ein entsprechendes Transportmedium, wobei für die Anwendung die tatsächliche Realisierung der Kommunikation verborgen bleibt. Die Kommunikationsebenen ORPC, OSEK COM, Gerätetreiber und Hardware bilden die Schichten 6 bis 1 des ISO/OSI-Referenzmodells, während Schicht 7, d. h. die Anwendungsschicht, direkt von der Client/Server-Anwendung bzw. den Anwendungsprotokollen übernommen wird.

Die Prozeßstruktur der Clients und Server beschreibt das Systemverhalten innerhalb der Architektur, womit allerdings im Unterschied zu konventionellen Client/Server-Systemen keine Interaktion mit der Außenwelt modelliert wird. Diese Schnittstelle zur Umwelt wird in der vorliegenden CSA durch zusätzliche Softwarebausteine, die sogenannte Firmware, gebildet. Diese Firmware trennt die hardwareabhängigen Teile der Funktion von der logischen Funktion des Client/Server-Systems. In ihr werden die Ein- und Ausgabeoperationen des Steuergerätes behandelt. Normalerweise sind dies die Ansteuerung von I/O-Pins des Prozessors mit definiertem Timing. Die Firmware wird hardwarenah speziell für das Steuergerät z. B. in Assembler oder C implementiert. Zur Client/Server-Umgebung stellt die Firmware eine Schnittstelle über Anwendungsprotokolle ähnlich denen von Client/Server-Prozessen bereit. Diejenigen Client- bzw. Server-Prozesse, die Aufträge aus der Firmware entgegennehmen oder diese beauftragen, werden als die primären Clients bzw. Server bezeichnet, so daß diese primären Prozesse direkt auf dem Steuergerät laufen, an dem auch die entsprechende Hardware angeschlossen ist. Über diese Zwischenschicht ist auch eine Kommunikation zu anderen als den Client/Server-Prozessen denkbar, indem für die betreffende Applikation eine Schnittstelle zur Client/Server-Umgebung in Form von entsprechender Firmware realisiert wird.

Zwischen zwei Client/Server-Prozessen findet die Kommunikation zum Datenaustausch über zwei asynchrone, unidirektionale Punkt-zu-Punkt-Kanäle statt. Sie basieren auf dem Prinzip des aus der Bürokommunikations-Datenverarbeitung bekannten Remote-Procedure-Call (RPC), wobei dieser Mechanismus für den Anwendungsfall von Kraftfahrzeugen auf die dort vorhandenen, begrenzten Ressourcen angepaßt und vereinfacht ist. So stehen z. B. keine Name-Services oder Security-Functions zur Verfügung, und es wird im allgemeinen auch keine sichere Kommunikation für den Nachrichtenaustausch vorausgesetzt. Der prinzipielle Ablauf des verwendeten OSEK-basierten Remote-Procedure-Call (ORPC) ist in Fig. 8 dargestellt. Bei einer Anforderung (Request) von einem Client- zu einem Server-Prozeß codiert eine Stub-Routine zunächst die Argumente in eine netzwerkneutrale, normalisierte Form. Der in der ORPC-Runtime-Bibliothek bereitgestellte Timed-RPC sorgt für die Übertragung des Requests über das Netzwerk oder über Kommunikationsobjekte im selben Adressraum zum Server. Dort decodiert die entsprechende Server-Stub-Routine die Argumente, ruft mit diesen Parametern die Dienstimplementierung auf und nimmt daraufhin die Ergebnisse entgegen. Diese werden wieder in die normalisierte Darstellung umgewandelt und zum Client zurückgesendet. Die Client-Stub-Routine sorgt für die Decodierung der Ergebnisse und gibt sie schließlich an den entsprechenden Prozeß zurück.

Für den Fall, daß bei der Übertragung über ein Medium zwischen verschiedenen Adressräumen eine Nachricht verloren geht, ist im ORPC ein Mechanismus implementiert,

der Nachrichten wiederholen kann. Dazu ist für jede Nachricht definiert, bis zu welchem Zeitpunkt nach dem Versenden des Requests die Antwort (Reply) erwartet wird. Kann diese Zeit nicht eingehalten werden, geht der Client davon aus, daß entweder seine Request-Nachricht oder der Reply darauf verloren ging. Er sendet deshalb den Request erneut aus. Zusätzlich ist eine Maximalanzahl für die Wiederholungen definiert, nach denen der Client mit einer entsprechenden Fehlermeldung über die Nichterfüllung des Dienstes benachrichtigt wird. Mit diesem Mechanismus können nur idempotente Dienste realisiert werden, bei denen eine Wiederholung eines bereits ausgeführten Dienstes keinen Einfluß auf das Gesamtergebnis oder den Systemzustand hat.

Im Unterschied zu herkömmlichen Implementierungen von RPC-Mechanismen für die Bürokommunikation ist bei der vorliegenden CSA weitaus mehr Funktionalität innerhalb der ORPC-Bibliotheksroutinen realisiert und standardisiert. Gängige RPC-Implementierungen stellen der Anwendung lediglich die benötigten Kommunikationsfunktionen in Form einer Bibliothek zur Verfügung und ermöglichen darüber hinaus nur die Generierung eines primitiven Gerüsts für die eigentliche Server-Funktionalität. Wenn dies, wie meist der Fall, nicht ausreicht, muß der Anwendungsprogrammierer den benötigten Server-Code selbst schreiben. Im Gegensatz dazu beinhaltet vorliegend die ORPC-Bibliothek neben den Kommunikationsroutinen auch den vollständigen Server-Code. Dieser wird von allen Client- und Server-Prozessen abgearbeitet, d. h. er wird pro Steuergerät genau einmal eingebunden. Dies erfüllt in vorteilhafter Weise die Anforderungen nach minimalem Ressourcenbedarf und maximaler Wiederverwendungsfähigkeit. Dieser Server-Code wird von der Anwendung lediglich mit den jeweiligen prozeßspezifischen Daten aufgerufen und arbeitet diese nach einem festgelegten Verfahren ab. Das zugehörige Server-Zustandsdiagramm ist in Fig. 9 dargestellt. Nach Aufruf durch die Anwendung durchläuft der Server eine Initialisierungsphase, die sich in vier einzelne Phasen gliedert, wie im zugehörigen Initialisierungsdiagramm in Fig. 10 dargestellt. Zunächst werden die anwendungsspezifischen Prozeßdaten initialisiert, wozu aus dem Server-Code heraus eine anwendungsspezifische Funktion aufgerufen wird, welche der Anwendungsprogrammierer zur Verfügung stellt. Dann erfolgt die Initialisierung der Kommunikationskanäle und der zugehörigen Datenobjekte. Der Server-Prozeß ist nun empfangsbereit und kann Anforderungen, d. h. Requests, von seinen Clients entgegennehmen. Der Prozeß geht daraufhin für eine einstellbare Zeit in eine Wartestellung, bis ein Request empfangen wird oder die eingestellte Zeit abgelaufen ist. Diese Maßnahme dient der Lastverteilung beim Anlauf des Systems bzw. dazu, unwichtigere Prozesse zu blockieren, bis sie tatsächlich benötigt werden. Zuletzt überprüft der Server, ob er alle notwendigen Initialisierungsinformationen hat. Ist dies nicht der Fall, fordert er sie selbsttätig, d. h. ohne Mitwirkung der Anwendung, bei den entsprechenden Clients an. Liegt nach einer einstellbaren Anzahl von Versuchen noch keine gültige Initialisierung vor, verzweigt der Server in eine Notlauf-Funktion, ansonsten betritt der Server eine Schleife, in der er die eingegangenen Anforderungen nacheinander abarbeitet.

Anstehende Requests werden dem Server-Prozeß über einen vom Betriebssystem bereitgestellten Signalisierungsmechanismus angezeigt. Anhand dieses Signals kann der Server zwischen verschiedenen Quellen unterscheiden und die entsprechende Bearbeitungsmethode anstoßen. Quelle dieser Signale können hierbei Nicht-CSA-Funktionen, wie Firmware, oder Betriebssystemfunktionen, wie Zeitgeber oder Interrupt-Mechanismen, oder andere CSA-Prozesse, wie Client-Prozesse, sein. Anhand des empfangenen Signals

wird vom Server-Code über eine Tabelle am entsprechenden SAP eine Stub-Routine aufgerufen. Die Stub-Routinen werden anhand der Informationen zu den Anwendungsprotokollen automatisch generiert und rufen die vom Anwendungsprogrammierer zur Verfügung gestellte Dienstimplementierung auf. Bei den Signalen wird zwischen sogenannten Application Events und den ORPC-Events unterschieden. Die Application Events werden zur Anbindung der Firmware und für Betriebssystemfunktionen genutzt und direkt in einen entsprechenden Aufruf umgesetzt. Die ORPC-Events sind durch den realisierten RPC-Mechanismus vorgegeben und dienen zur Anbindung anderer CSA-Prozesse.

Für die Realisierung des RPC kommen drei Varianten in Betracht, von denen der gebräuchlichste ein sogenannter synchroner RPC ist, der durch die Funktion Timed-RPC der ORPC-Bibliothek realisiert ist. **Fig. 11** zeigt das Flußdiagramm dieser Funktion. Nach der Initialisierung der lokalen Daten der Funktion werden in einer Schleife die für diese Methode konfigurierten Aufrufversuche (Attempts) abgearbeitet. Bei jedem Versuch wird hierzu zunächst ein Zeitgeber mit dem entsprechenden Timeout-Wert geladen. Im Anschluß daran wird der Request versendet und auf den Ausgang dieser Sende-Operation gewartet. War das Senden des Requests erfolgreich, betritt die Funktion eine Schleife, in der sie auf die Antwort, d. h. den Reply, wartet. Wird die Antwort nicht innerhalb der vorgegebenen Zeitspanne empfangen, startet ein erneuter Übertragungsversuch, sofern die Anzahl möglicher Versuche noch nicht überschritten ist. Läuft der Zeitgeber ab, bevor ein erfolgreicher Ausgang der Sendeoperation signalisiert wurde, wird ohne Warten auf Antwort der nächste Versuch gestartet. Auf Server-Seite wird durch den Empfang eines Requests eine entsprechende Server-Stub-Routine aufgerufen, welche ihrerseits die eigentliche Dienstimplementierung aufruft und nach deren Abarbeitung eine entsprechende Antwort an den Client-Prozeß zurücksendet. Abhängig vom Ausgang liefert die Funktion einen entsprechenden Fehlercode sowie im Erfolgsfall das zugehörige Ergebnis an die aufrufende Funktion, in diesem Fall eine Client-Stub-Routine, zurück. Innerhalb der Anwendung muß bei einem RPC zunächst dieser Fehlercode betrachtet werden, bevor im Erfolgsfall mit dem zurückgelieferten Ergebnis gearbeitet werden kann. Für den Fall, daß ein RPC-Vorgang fehlschlägt, ist eine entsprechende Fehlerbehandlung vom Anwendungsprogrammierer vorzusehen.

Als weitere Variante kann ein sogenannter asynchroner RPC vorgesehen sein. Dieser entspricht einer Modifikation des synchronen RPC dahingehend, daß von der Server-Stub-Routine eine Antwort gesendet wird, bevor die eigentliche Dienstimplementierung aufgerufen wird. Der Client-Prozeß kann dadurch asynchron zu dem von ihm beauftragten Server-Prozeß weiterarbeiten. Der Server-Prozeß bzw. die aufgerufene Stub-Routine bestätigt hierbei durch den Reply lediglich den korrekten Empfang des entsprechenden Requests, nicht jedoch die erfolgte Bearbeitung. Infolgedessen ist diese Art des RPC nur für Methoden zulässig, die kein Ergebnis, d. h. einen Rückgabewert vom Typ "void", haben. Die Unterscheidung zwischen synchronem und asynchronem RPC erfolgt allein in der Server-Stub-Routine, während die verwendeten Client-Stub-Routinen in beiden Fällen identisch sind. Auch wird in beiden Fällen zur Kommunikation die Funktion Timed-RPC aufgerufen.

Als dritte Variante kann der sogenannte Oneway-RPC vorgesehen sein, bei welchem im Gegensatz zu den beiden vorigen Mechanismen keine Antwort vom Server an den Client generiert wird. Dieser Mechanismus ist ebenfalls nur für Funktionen zulässig, die kein Ergebnis liefern. Im Unterschied zu Timed-RPC realisiert die verwendete Kommuni-

kationsfunktion Oneway-RPC keinen Timeout-Retransmit-Mechanismus und wartet nicht auf eine Antwort des Servers. Das zu dieser RPC-Variante gehörige Flußdiagramm ist in **Fig. 12** dargestellt.

Neben dem Aufruf der entsprechenden Kommunikationsfunktion bzw. Dienstimplementierung sind die Client- und Server-Stub-Routinen auch für das sogenannte Marshalling, d. h. die Konvertierung eines einfachen Datentyps aus einer prozessorspezifischen Darstellung in das zur Kommunikation verwendete normalisierte Format, und Unmarshalling, d. h. der Datenkonvertierung vom normalisierten Format in die prozessorabhängige Darstellung, der zugehörigen Parameter und Rückgabewerte verantwortlich. Dabei werden Byte- und Bit-Ordering und die Länge der Darstellung berücksichtigt. Die Client- und Server-Stub-Routinen werden anhand der Signatur einer Methode, d. h. unter Berücksichtigung von Anzahl, Reihenfolge und Typ der Parameter sowie des Rückgabewertes, automatisch generiert. Die Stub-Routinen enthalten die entsprechende Aufruf-Sequenz von Konvertierungsroutinen für einfache Datentypen. Während bei den herkömmlichen Realisierungen der Anwendungsprogrammierer selbst Funktionen für das Marshalling bzw. Unmarshalling bereitstellen und an die Stub-Routine übergeben muß, wird diese Funktionalität hierdurch beim vorliegenden System vollständig innerhalb der Stubs gekapselt und bleibt der Anwendung verborgen. Dadurch ist der in der vorliegenden CSA realisierte RPC bezüglich der Schnittstelle für den Anwendungsprogrammierer transparent, d. h. durch die gewählte Vorgehensweise ist die Schnittstelle der Anwendung zu den Stub-Routinen so gestaltet, daß diese exakt derjenigen bei einem lokalen Funktionsaufruf entspricht. Für jede Signatur existiert genau ein Paar von Stub-Routinen, wobei Methoden mit gleicher Signatur dieselben Stub-Routinen verwenden. Dies ist von der methodenspezifischen Generierung von Stub-Routinen herkömmlicher Realisierungen zu unterscheiden. Die vorliegende CSA macht im Unterschied zu herkömmlichen Architekturen von der Möglichkeit einer (Wieder-)Verwendung von Stub-Routinen durch mehrere Methoden Gebrauch.

Eine Randbedingung für die normalisierte Darstellung in einem Netzwerk ist, daß auf den eingesetzten Microcontrollern der Aufwand für die Umsetzung möglichst gering ist. Anstelle herkömmlicher Realisierungen einer External-Data-Representation wurde daher für das vorliegende System eine eigenständige normalisierte Darstellung in Form einer OXDR (OSEK-basierte externe Datenrepräsentation) definiert. Konvertierungsroutinen für einfache Datentypen werden in einer Bibliothek zur Verfügung gestellt. OXDR zeichnet sich durch eine Trennung von Marshalling- und Unmarshalling-Routinen aus, was selektives Einbinden der benötigten Module und somit minimalen ROM-Bedarf ermöglicht. Außerdem können für die Datenkonvertierung Quell- und Zieldatenbereich identisch sein, was den RAM-Bedarf der Konvertierungsroutinen minimiert.

Für die Protokollimplementierung erhält jede Methode eines Anwendungsprotokolls eine eindeutige, als Dienstnummer bezeichnete Nummer. Mit dieser identifiziert sowohl der Client den gewünschten Dienst als auch der Server die dazugehörige Dienstimplementierung. In den Nachrichten sind in den Nutzdaten sowohl der Nachrichtentyp, d. h. Request, Reply oder Error, als auch die Dienstnummer und die übergebenen Daten kodiert. **Fig. 13** zeigt eine Darstellung des so realisierten Protokolls. Dabei bedeuten Bit 7 ein Error-Flag, Bit 6 Reply-Flag und die Bits 5 bis 0 die Service-ID zwischen 0 bis 63. Die Anfangsbits "00 . . ." bedeuten einen Request für den Dienst mit der anschließenden Nummer, die Anfangsbits "01 . . ." einen Reply auf den Dienstauftrag mit der anschließenden Nummer und die Anfangsbits

"11. . ." einen Fehler bei der Verarbeitung des Dienstes mit der anschließenden Nummer. Mit Application-Data sind die beim Aufruf der Methode übergebenen Daten bzw. die Antwort darauf bezeichnet. Der Client sendet also eine Nachricht mit einer Dienstnummer an seinen Server, der anhand der gelöschten Bits 6 und 7 und der Dienstnummer prüft, ob es sich um einen Request handelt und ob er diesen Dienst erbringen kann. Nach der erfolgreichen Verarbeitung des Dienstes sendet der Server die Ergebnisse unter derselben Dienstnummer mit dem gesetzten Bit 6 an den Client zurück. Dieser erkennt an der Dienstnummer, dem gesetzten Bit 6 und dem gelöschten Bit 7, daß es sich um ein gültiges Ergebnis zum aufgerufenen Dienst handelt. Falls der Server einen geforderten Dienst nicht erbringen kann, beantwortet er den Request mit einer Fehlernachricht, indem er die Bit 6 und 7 setzt und in den Anwendungsdaten den entsprechenden Fehler kodiert.

Im Ergebnis steht somit für das Steuerungssystem ein objektorientierter Ansatz zur Verfügung, bei der alle in der CSA verwendeten Prozesse eine oder mehrere definierte Schnittstellen in Form von SAPs bzw. SAPIFs haben, über die mit Ihnen mittels Anwendungsprotokollen kommuniziert wird. Die Prozesse können lokale Daten besitzen, die nur über die Anwendungsprotokolle modifiziert werden können, und sie besitzen einen inneren Zustand, da das Verhalten eines solchen Prozesses über einen einfachen Automaten (FSM) festgelegt ist. Damit folgt der Entwurf mit der Client/Server-Methode den wesentlichen Paradigmen des objektorientierten Designs. Das Design der Anwendung erfolgt auf Klassenebene durch Aufbau von Kommunikationsbeziehungen zwischen den Prozeßklassen. Zur Generierungszeit werden aus diesen kommunizierenden Prozeßklassen die Prozeßinstanzen erzeugt und mit entsprechenden Daten vorbelegt, wobei auch Polymorphismus durch Überschreiben der Dienstimplementierungen an dieser Stelle möglich ist. Als hierarchische Strukturierungsmittel finden Frames Verwendung, die andere Frames oder Prozeßklassen, Firmware und Kommunikationsverbindungen enthalten können, sogenannte Construction of Parts. Aus diesen Frames kann dann nach der Bottomup-Designmethode, d. h. durch Construction from Parts, die Anwendung zusammengebaut werden. Ein einmal angelegtes, implementiertes und getestetes Frame läßt sich in beliebig vielen Anwendungen wiederverwenden, so daß eine erhebliche Effizienz- und Effektivitätssteigerung gegenüber dem konventionellen Entwurf ermöglicht wird. Durch die vorliegend realisierte CSA ist eine hohe Flexibilität auch im Bereich der Kraftfahrzeugelektronik gegeben, die eine baureihenübergreifende Verwendung der Anwendungsfunktionen oder zumindest von Teilen derselben erlaubt. Wird beispielsweise bei einer neuen Baureihe eine Anwendungsfunktion nur hinsichtlich ihrer Funktionslogik, nicht aber ihrer zugehörigen Sensorik oder Aktuatorik verändert, reicht eine Modifizierung des entsprechenden Funktionsmonitors aus. Durch die vorliegende Strukturierung von Funktionen wird folglich auch die Wartungsfähigkeit verbessert.

Der Entwicklungsprozeß wird formal durch ein Design- und Implementierungswerkzeug unterstützt, mit dem der gesamte Desingprozeß und Teile des Implementierungsprozesses unterstützt werden. Damit können in der Designphase die Prozeßklassen mit ihren Schnittstellen und Daten spezifiziert werden. Zusammen mit den entwickelten Anwendungsprotokollen lassen sie sich zu Frames kombinieren, die ihrerseits in anderen Frames genutzt werden können. Am Schluß des Design-Prozesses für eine neue Funktion wird der Kontext festgelegt, in welchem die Frames angewendet werden, z. B. zur Festlegung der Frequenz und des Tastverhältnisses im Fall der Blinkfunktion eines Fahr-

zeugblinkgebers. Aus den mit Hilfe des Entwicklungswerkzeugs entworfenen Funktionen wird eine Parts-Bibliothek aufgebaut.

Wie die vorstehende Beschreibung eines Anwendungsfalls für ein Kraftfahrzeug zeigt, bietet das erfindungsgemäße Steuerungssystem mit der implementierten Client/Server-Architektur erhebliche Vorteile. Der Entwicklungsaufwand wird durch systematische Wiederverwendung und Änderungsfreundlichkeit sowie weitgehende Automatisierung vereinfacht, wozu eine klare Trennung zwischen logischem Entwurf einer Fahrzeugfunktion und deren physikalischer Einbettung in die Steuergerätestruktur realisiert ist. Außer Funktionsentwürfen sind auch entsprechende Simulationsmodelle und implementierte Software-Module in gleicher Weise flexibel einsetzbar, was den Aufwand nicht nur in der Entwurfsphase, sondern auch in der Implementierungs- und in der Integrationsphase verringert. Durch die erfindungsgemäße Systemauslegung ergibt sich ein großer Freiheitsgrad bezüglich der Platzierung einzelner Funktionen auf den Steuergeräten im vernetzten Steuergeräteverbund.

Wenngleich die Erfindung im Detail anhand eines Kraftfahrzeug-Steuerungssystems erläutert wurde, versteht es sich, daß sie auch auf andere Arten von datenverarbeitungsgestützten elektronischen Steuerungssystemen mit einem Steuergeräteverbund aus verteilt angeordneten Steuergeräten anwendbar ist.

Patentansprüche

1. Datenverarbeitungsgestütztes elektronisches Steuerungssystem, insbesondere für ein Kraftfahrzeug, mit
 - einem Anwendungsfunktionen ausführenden Steuergeräteverbund mit mehreren, verteilt angeordneten Steuergeräten (1a, 1b, 1c) und einem diese verbindenden Datenübertragungsnetzwerk (2), **dadurch gekennzeichnet**, daß
 - die Anwendungsfunktionen in Form einer Client/Server-Architektur in dem Steuergeräteverbund implementiert sind.
2. Steuerungssystem nach Anspruch 1, weiter dadurch gekennzeichnet, daß die Client/Server-Architektur für eine jeweilige Anwendungsfunktion eine Client-Ebene (5), eine Server-Ebene (6) und eine zwischenliegenden Funktionsmonitorebene (7) beinhaltet, welche Dienst-anforderungen von der Client-Ebene und/oder von übergeordneten Anwendungsfunktionen empfängt und unter Benutzung von Diensten der Server-Ebene und/oder von untergeordneten Anwendungsfunktionen bearbeitet.
3. Steuerungssystem nach Anspruch 2, weiter dadurch gekennzeichnet, daß
 - die Client-Ebene (5) wenigstens einen primären Client (5a) und einen zugehörigen Requestor (5b) enthält, wobei der Requestor ereignisauslösende Hardware-Einheiten und eine zugehörige Steuergeräte-Firmware repräsentiert und der primäre Client den Requestor verwaltet, Diensteanforderungen von ihm entgegennimmt und Aufträge an die Funktionsmonitorebene (7) absetzt, und/oder
 - die Funktionsmonitorebene pro Anwendungsfunktion einen Funktionsmonitor (7a), der Dienst-anforderungen der Client-Ebene (5) entgegen-nimmt und bearbeitet, sowie diesem untergeordnete Monitore (7b) zur Verwaltung von Teilfunktionen enthält und/oder
 - die Server-Ebene (6) wenigstens einen primären Server (6a) und einen zugehörigen Füller

- (6b) beinhaltet, wobei die Füller ausführende Hardware-Einheiten und zugehörige Steuergeräte-Firmware repräsentieren und der primäre Server den Füller verwaltet und mit der Ausführung von Diensten beauftragt sowie Dienstanforderungen von der Funktionsmonitorebene (7) entgegennimmt. 5
4. Steuerungssystem nach einem der Ansprüche 1 bis 3, weiter dadurch gekennzeichnet, daß als eine Gruppe von Designelementen für den Funktionsentwurf der Client/Server-Architektur Service-Access-Points (SAPs) vorgesehen sind, die Anwendungsprozeß-Schnittstellen auf dem Niveau von Schicht 7 des ISO/OSI-Referenzmodells bilden und je ein Protokoll in Client- und Serverrolle beinhalten. 10 15
5. Steuerungssystem nach einem der Ansprüche 1 bis 4, weiter dadurch gekennzeichnet, daß als eine Gruppe von Designelementen für den Funktionsentwurf der Client/Server-Architektur Ports als horizontale Kommunikationsschnittstellen auf Schicht 7 des ISO/OSI-Referenzmodells als Ankerpunkte für die bidirektionale Client/Server-Kommunikationsverbindung zur Implementierungszeit vorgesehen sind. 20
6. Steuerungssystem nach einem der Ansprüche 1 bis 5, weiter dadurch gekennzeichnet, daß als eine Gruppe von Designelementen für den Funktionsentwurf der Client/Server-Architektur Prozesse in Form von Prozeßklassen vorgesehen sind, die eine äußere Schnittstelle, eine innere Struktur und ein Verhalten umfassen, welches Änderungen des internen Zustands (z) der Prozeßklasse, Modifikationen an gekapselten Datenelementen und die Ausführung von Aktionen beinhaltet. 25 30
7. Steuerungssystem nach einem der Ansprüche 1 bis 6, weiter dadurch gekennzeichnet, daß in den Steuergeräten eine Betriebssystemschicht mit einem echtzeitfähigen Multitasking-Betriebssystem implementiert ist und als Kommunikationsschicht eine solche vom Remote-Procedure-Call-Typ (RPC) verwendet ist, wobei die Client/Server-Prozesse auf die Dienste des Betriebssystems und der Kommunikationsschicht ohne direkten Hardware-Zugriff zurückgreifen. 35 40
8. Steuerungssystem nach Anspruch 7, weiter dadurch gekennzeichnet, daß in einer RPC-Bibliothek ein vollständiger Server-Code abgelegt ist, der pro Steuergerät genau einmal eingebunden ist und von allen Client- und Server-Prozessen abgearbeitet wird. 45
9. Steuerungssystem nach Anspruch 7 oder 8, weiter dadurch gekennzeichnet, daß der RPC-Vorgang für die Kommunikationsschicht als synchroner, asynchroner oder Oneway-RPC-Vorgang realisiert ist. 50
10. Steuerungssystem nach einem der Ansprüche 7 bis 9, weiter dadurch gekennzeichnet, daß das Nachrichtenprotokoll Informationen über den Nachrichtentyp, eine Dienstnummer einer jeweiligen Methode eines Anwendungsprotokolls und die zu übergebenden Daten enthält. 55

Hierzu 8 Seite(n) Zeichnungen

- Leerseite -

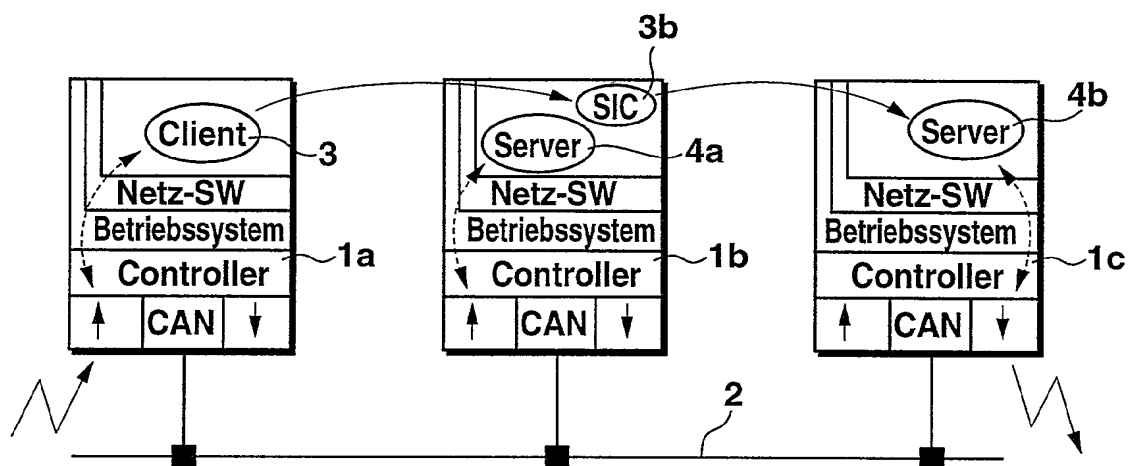
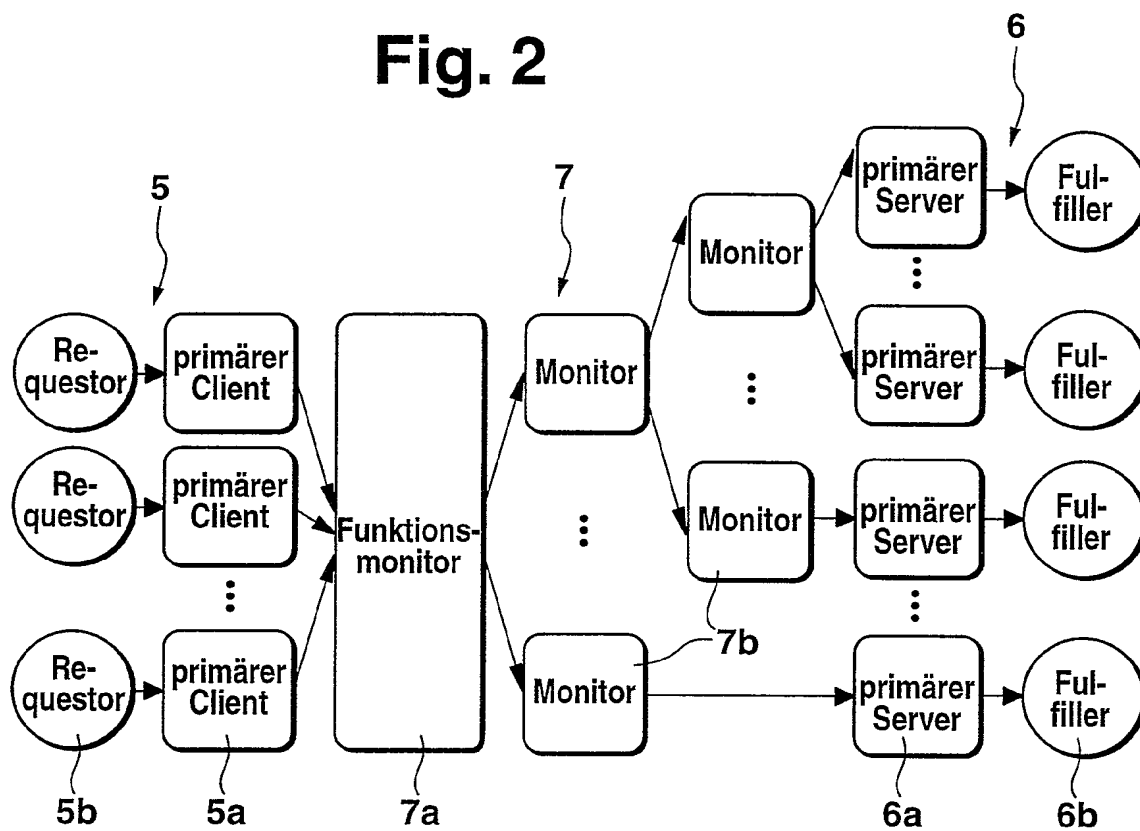
Fig. 1**Fig. 2**

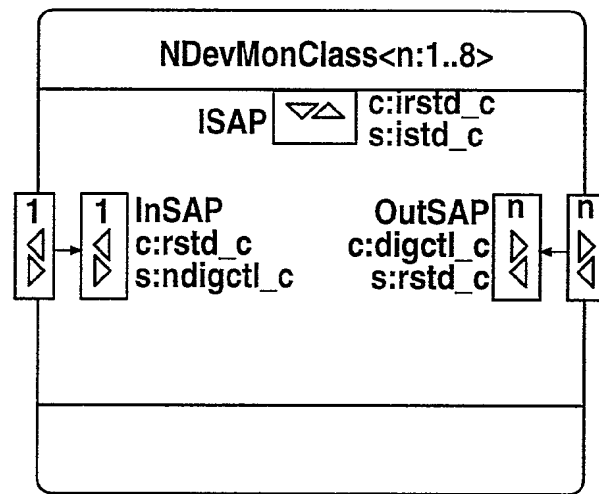
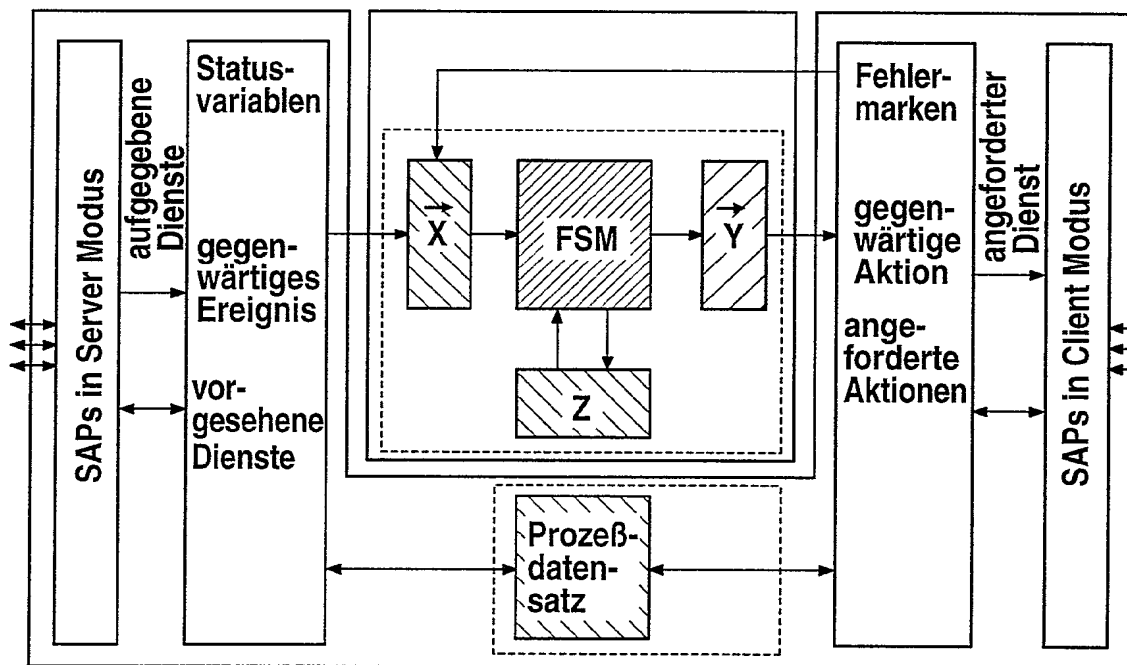
Fig. 3**Fig. 4**

Fig. 5

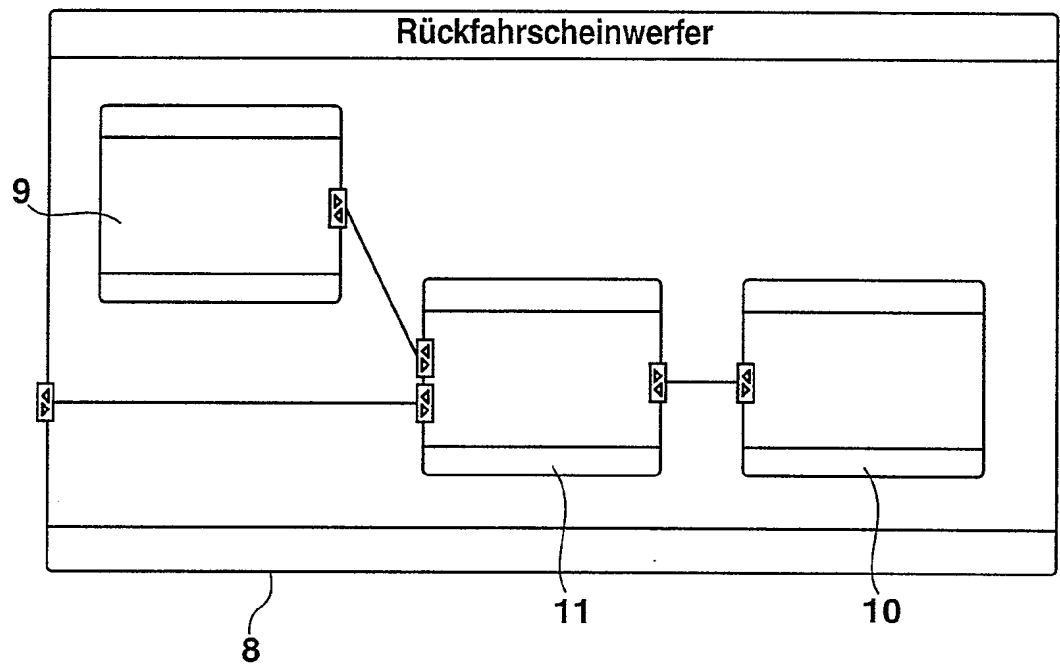


Fig. 6

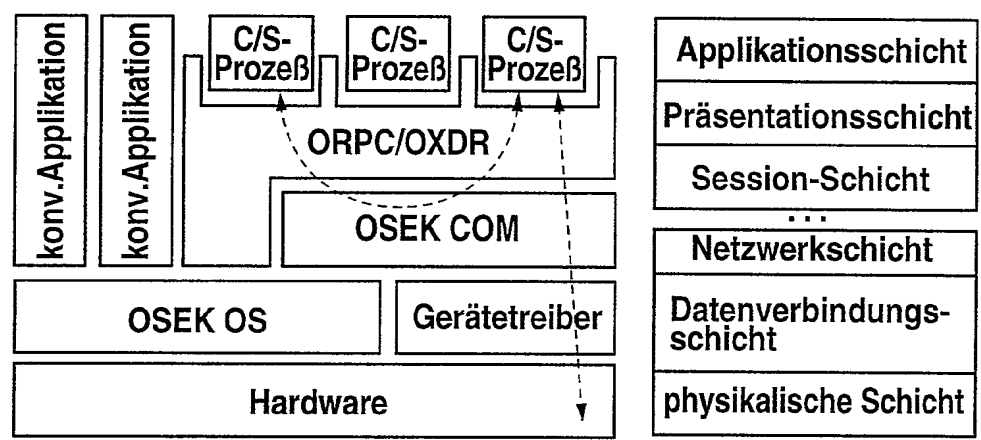
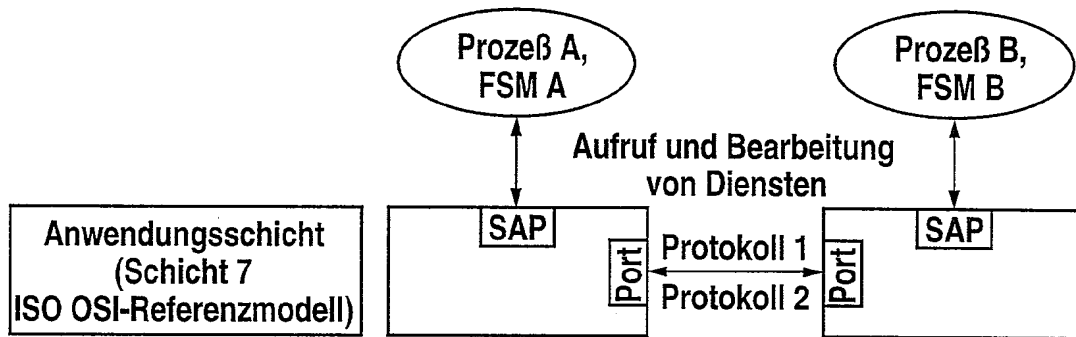
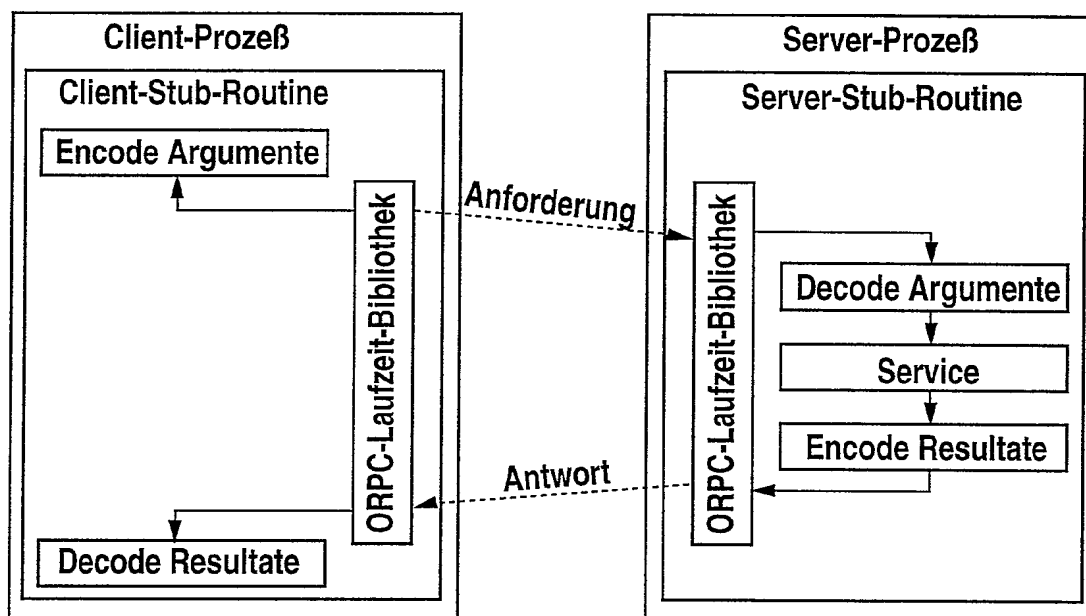


Fig. 7**Fig. 8**

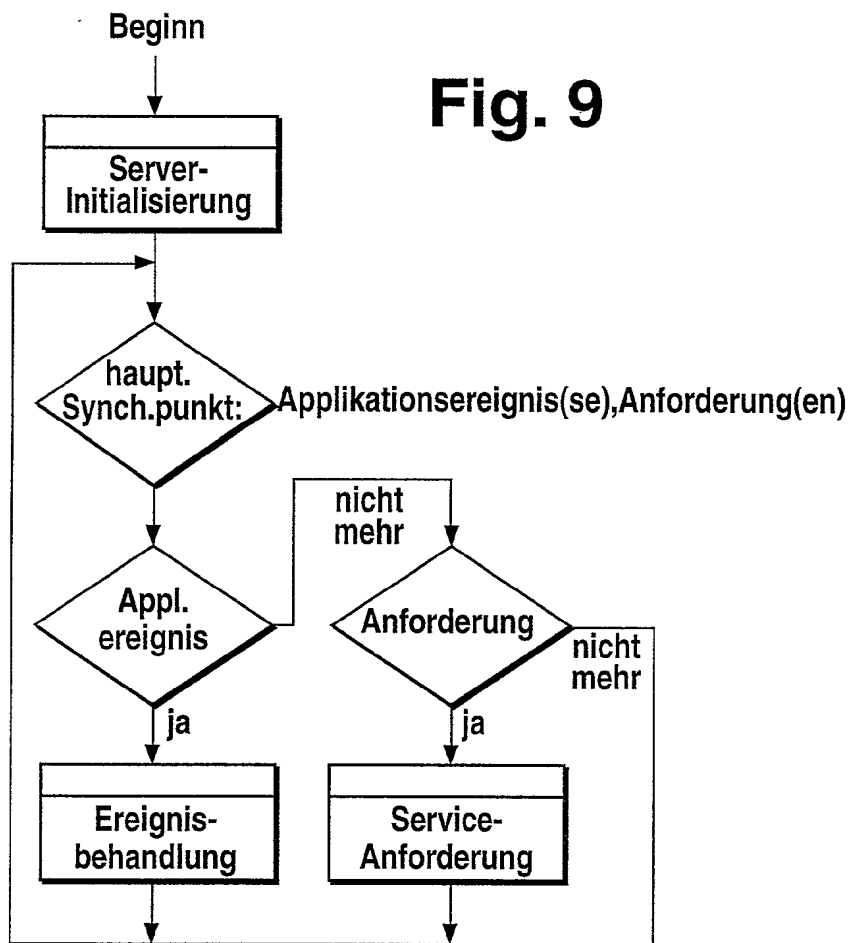
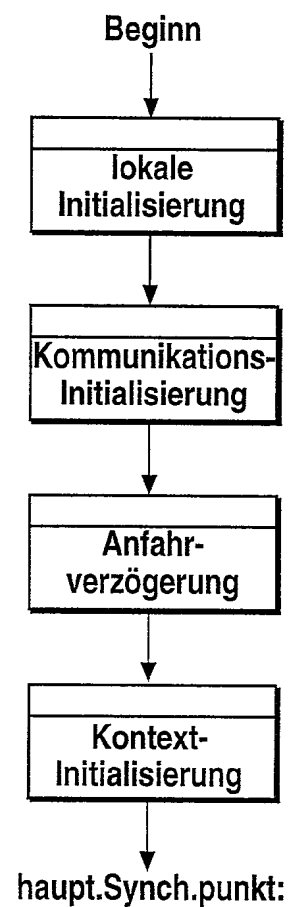
**Fig. 10**

Fig. 11

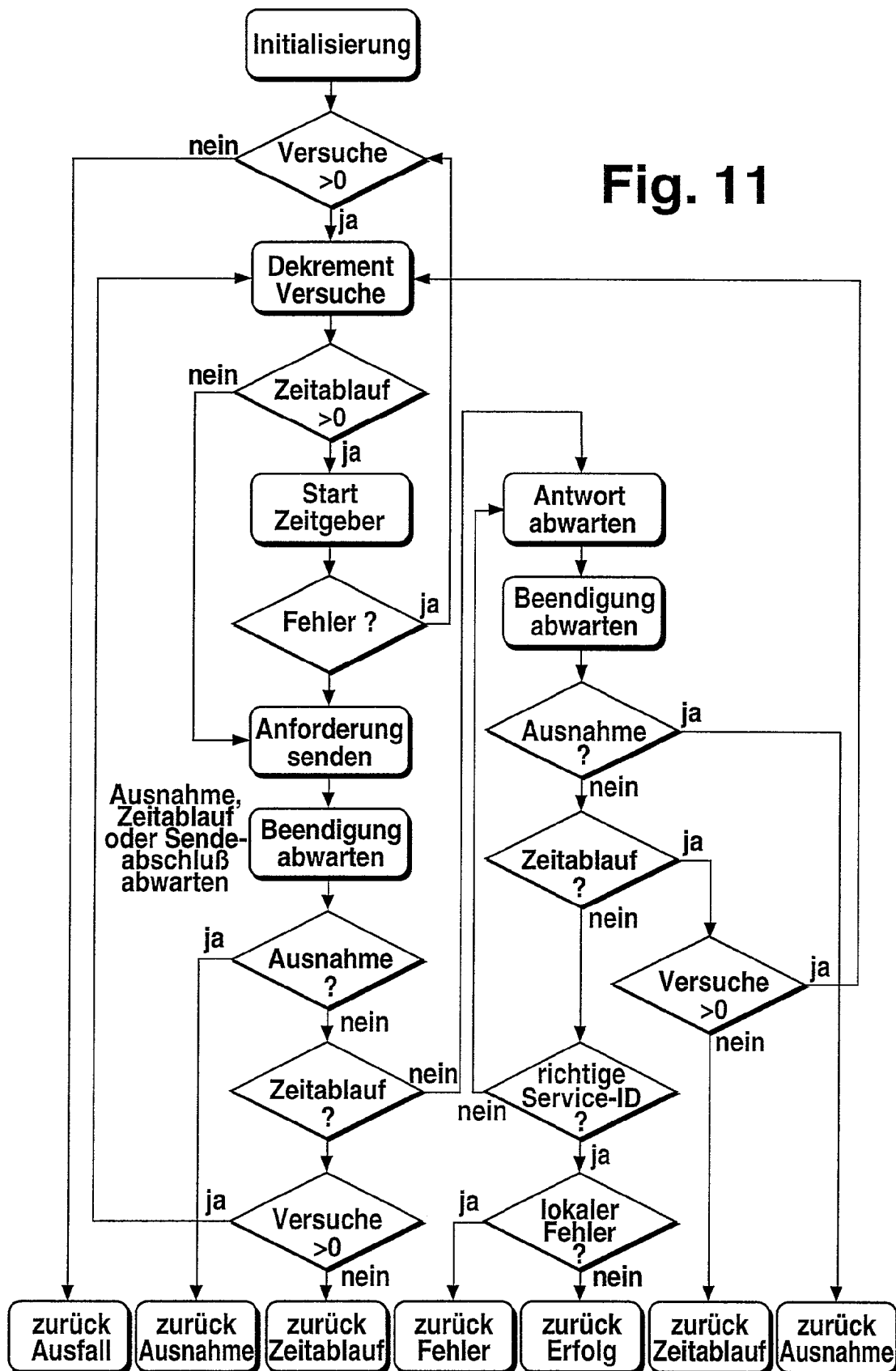


Fig. 12

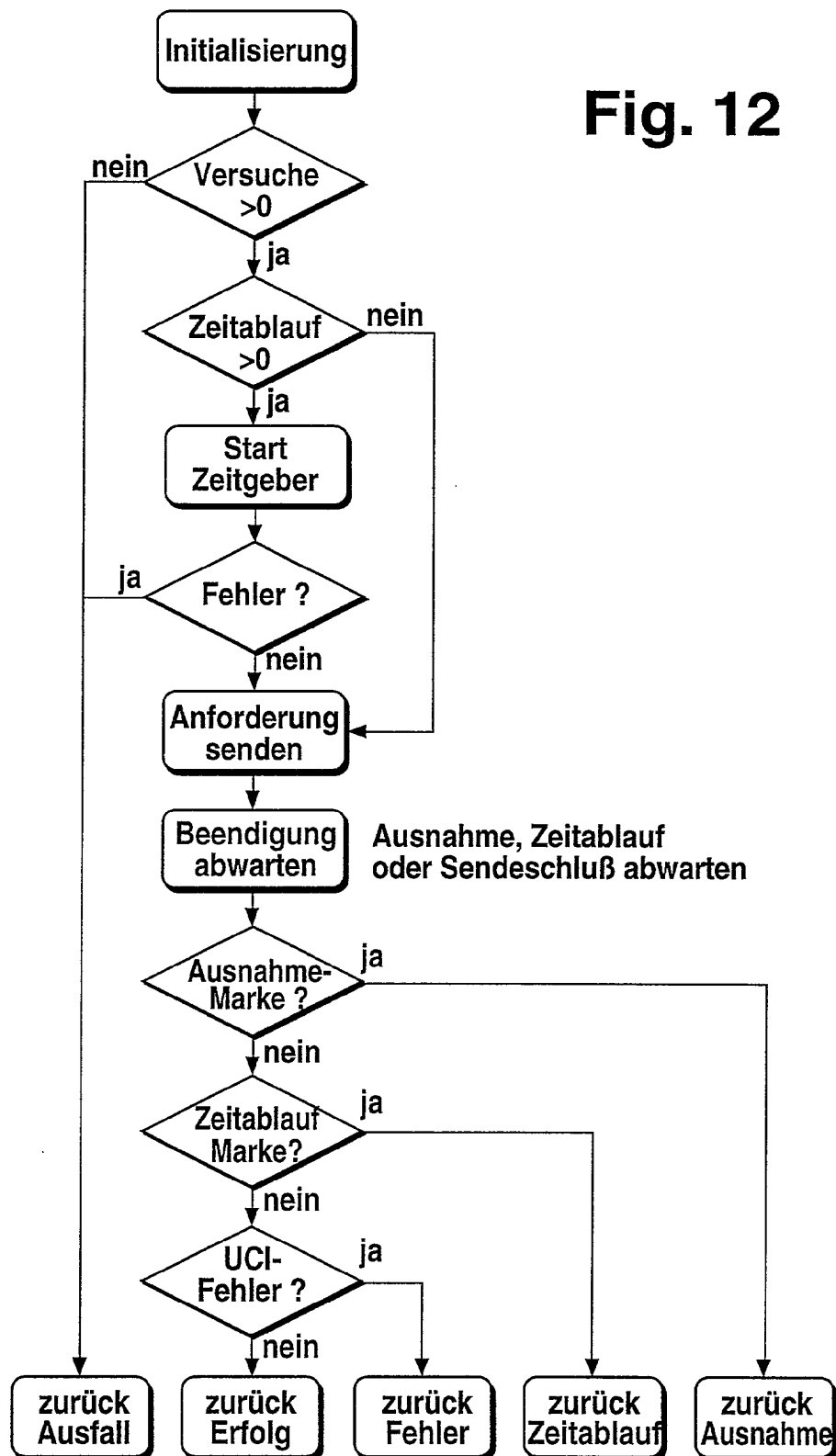


Fig. 13